VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 David Michalík

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

WEBOVÁ APLIKACE V PLC PFC200
RICH INTERNET APPLICATION IN PLC PFC200

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — David Michalík
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jakub Arm
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: David Michalík ***ID:*** 173706
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Webová aplikace v PLC PFC200

POKYNY PRO VYPRACOVÁNÍ:

1) Seznamte se s embedded Linux prostředím v PLC PFC200 firmy WAGO.
2) Proveďte rešerši volně dostupných webových serverů v jazyce C/C++ pro Linux.
3) Prostudujte možnosti hardwarové abstraktní vrstvy DAL a jejího rozhraní ADI.
4) Vytvořte internetovou aplikaci (pomocí technologií PHP, JavaScript, HTML5) běžící na zvoleném serveru v PLC, aby bylo možné načítat a ukládat hodnoty PLC (vstupy, výstupy, vnitřní M paměť) přes DAL vrstvu PLC.
5) Vytvořenou aplikaci otestujte a vyhodnoťte její funkčnost.

DOPORUČENÁ LITERATURA:

Bruce Eckel. Myslíme v jazyku C++. Grada, 2000. ISBN 80-247-9009-2.

Thau Dave. Velký průvodce JavaScriptem. ISBN 978-80-247-2211-5.

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 29.5.2017

Vedoucí práce: Ing. Jakub Arm
Konzultant: Jakub Svoboda

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato bakalářská práce se věnuje implementaci vlastního způsobu ovládání vstupů, výstupů a paměti PLC PFC200 od firmy WAGO pomocí dynamického webového rozhraní. Součástí práce je popis PLC, způsobů jeho ovládání, rešerše volně dostupných webových serverů pro Linux a popis metod pro vytvoření real-time ovládání. Výstupem je webová aplikace běžící na webovém serveru v PLC, využívající vlastní runtime systém formou daemon programů.

KLÍČOVÁ SLOVA

ADI, daemon, DAL, embedded systémy, Lighttpd, Linux, NVRAM, PFC200, PHP, PLC, runtime systém, WAGO, webová stránka, webový server

ABSTRACT

This bachelor's thesis is mainly focused on implementation of our own means of controlling a PLC PFC200, made by Wago company, through a dynamic website application. This thesis includes basic information about PLC, known means of controlling it, a list of available web servers for Linux operating systems and a description of how to create our own real-time control. Result of this thesis is a web application running on a webserver inside the PLC using our own runtime system in a form of daemon programs.

KEYWORDS

ADI, daemon, DAL, embedded systems, Lighttpd, Linux, NVRAM, PFC200, PHP, PLC, runtime system, WAGO, website, webserver

MICHALÍK, David *Webová aplikace v PLC PFC200*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2017. 52 s. Vedoucí práce byl Ing. Jakub Arm

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Webová aplikace v PLC PFC200“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu své semestrální práce, panu Ing. Jakubovi Armovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno . . . .

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ

ÚVOD

Účelem této bakalářské práce je vytvořit jednoduchou a uživatelsky přátelskou vizualizaci ovládaní PLC PFC200 750-8202 od firmy WAGO prostřednictvím dynamických webových stránek využívající JavaScript, PHP a HTML5. Přes toto rozhraní bude uživatel schopen operovat se vstupními a výstupními moduly připojených na PLC a vyčítat/zapisovat do NVRAM v reálném čase.

Abychom mohli tohoto cíle dosáhnout, je potřeba se obeznámit s embedded Linux prostředím v PLC - Pengutronixu, jelikož právě do tohoto prostředí budeme implementovat volně dostupný webový server a výsledné řešení. Proto je nutné provést rešerši nejvyužívanějších webových serverů pro Linux a vybrat z nich ten nejvhodnější v závislosti na jeho parametrech. Tyto servery se liší například v náročnosti na paměť nebo procesorový výkon.

Nadále se v teoretické části této bakalářské práce budeme věnovat technickému popisu PLC, využití vrstvy ADI/DAL ke komunikaci s I/O moduly, knihovně NVRAM pro zápis a čtení z non-volatile paměti a popisu nástrojů pro vytvoření dynamické webové aplikace - hlavně tedy jejich celkovému konceptu využití - HTML, JavaScriptu a PHP.

V praktické části jsou poté popsány kroky, které jsou potřeba ke splnění zadání práce. Důležitou částí je správná instalace embedded systému, pochopení fungování webového serveru Lighttpd a výsledný návrh řešení, který v sobě obsahuje dynamické metody webové aplikace operující s JSON soubory, ke kterým přistupuje námi vytvořený daemon runtime systém a ten následně komunikuje s hardwarovými prvky pomocí ADI/DAL, případně knihovny nvram.h.

1 TEORETICKÁ ČÁST

1.1 Wago PLC PFC200

1.1.1 Základní specifikace

PFC200 CS 2ETH RS, konkrétně model 750-8202 spadá pod automatizační zařízení, které dokážou vykonávat ovládací úlohy PLC - Programmable Logic Controler. Využívají se v průmyslových aplikacích pro zjednodušení a automatizování výrobních procesů. Zjednodušení proto, že v současné době se přechází na nový trend Internet of Things a Industry 4.0, jehož cílem je tyto systémy modulárně propojit, takže se vytratí nutnost programování na nízké úrovni a technici se mohou zaměřit na obecný, globální pohled a vymýšlet nové způsoby řešení problémů.

Automatizační úlohy je možno psát ve všech programových jazycích, které jsou součástí IEC 61131-3 normy:

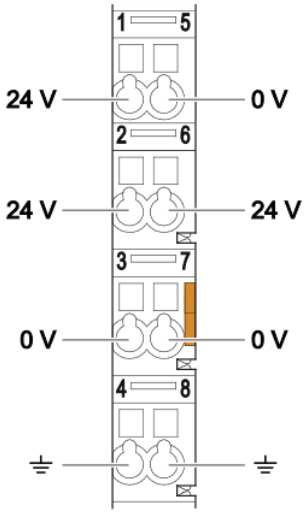
Programovací jazyky IEC 61131-3

- IL - Intruction List
- ST - Structured List
- LD - Ladder Diagram
- FBD - Function Block Diagram
- SFC - Sequential Function Chart
- CFC - Continuous Function Chart

V závislosti na preferencích programátora je možno v těchto jazycích programovat ve vývojovém prostředí CODESYS (COntrolled DEvelopment SYStem) nebo v novějším e!COCKPIT. Obě tato vývojová prostředí pracují ve svém specifickém runtime-u, který je spuštěn jako proces na pozadí Pengutronixu (CODESYS2 nebo e!RUNTIME). V PFC200 tedy na pozadí operuje průmyslový embedded linux systém - Pengutronix, který otevírá programátorům nové možnosti ovládaní. Ať už se uživatel vyzná v jazycích normy IEC 61131-3 nebo v Linuxovém prostředí, může naplno využít možnosti našeho PLC.

K PFC200 je možné připojit všechny možné vstupní a výstupní moduly WAGO-I/O-SYSTEM série 750 a 753. Moduly se připojují jako karty za sebou a jsou uzavřeny zakončovacím modulem. Je možno standardně zapojit 64 modulů a s rozšířením sběrnice až na 250.

I/O moduly od společnosti WAGO obsahují licencované CAGE CLAMP připojovací svorky.



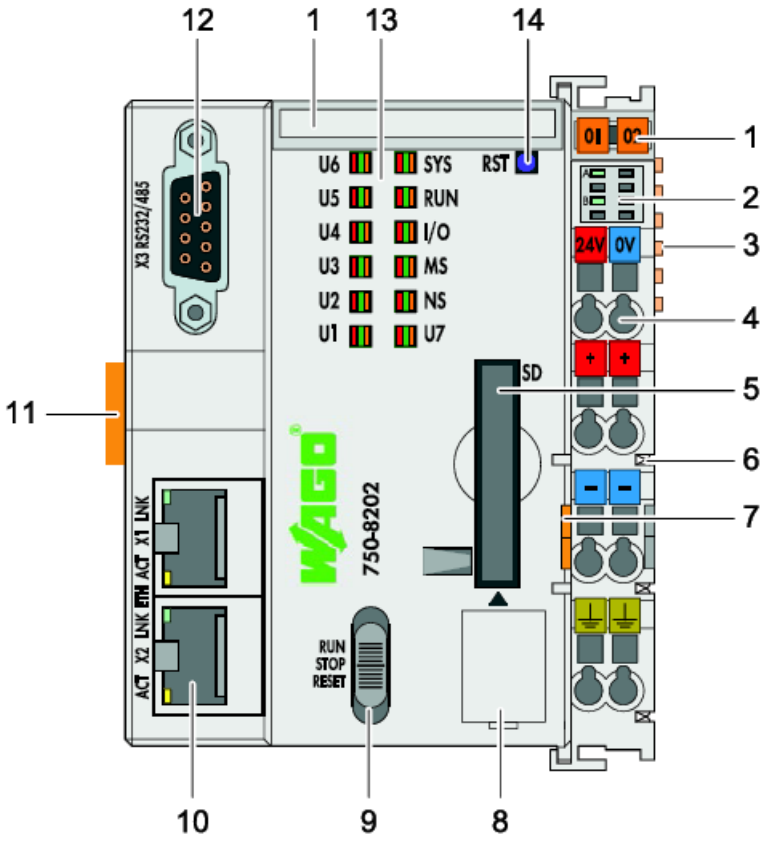
Obr. 1.1: CAGE CLAMP I/O modul [1]

1.1.2 Technické parametry

V následující tabulce jsou vypsány pouze některé ze základních parametrů PFC200, zbytek je možný k nalezení v oficiálním datasheetu.

CPU	Cortex A8, 600MHz
Operační systém	Real-time Linux
Hlavní paměť (RAM)	256 MB
Vnitřní flash paměť	256 MB
Retain paměť (NVRAM)	128 KB
ETHERNET	2 x RJ-45 (přehozené)
Sériové rozhraní	RS-232/-485 (přepínatelné)
Protokoly	DHCP, DNS, NTP, FTP, FTPS, SNMP, HTTP, HTTPS, SSH, MODBUS (TCP, UDP, RTU)
Operační teplota	0°C ... 55°C

Tab. 1.1: Technické parametry PFC200 750-8202



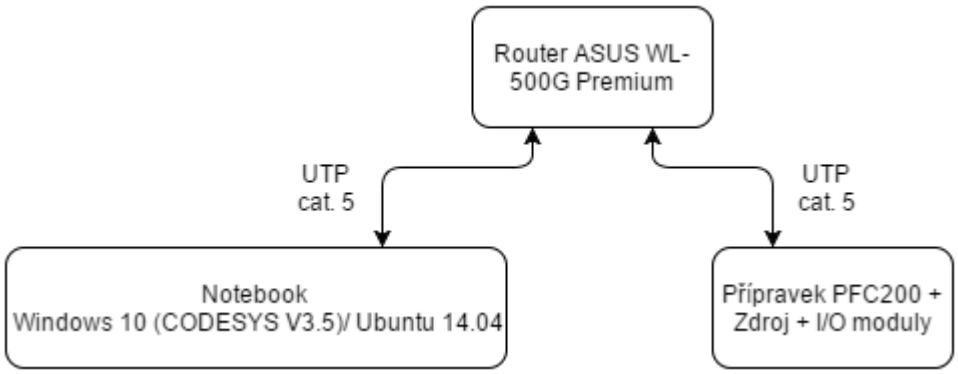
Obr. 1.2: Wago PLC PFC200 - popis [1]

Popis Wago PFC200

1. Volba označení
2. LED indikátory napájení
3. Přípojnice modulů
4. Připojení napájení CAGE CLAMP
5. Slot pro SD kartu
6. Kontakty pro napájení I/O modulů
7. Pásek pro vyjmutí modulů
8. Servisní rozhraní
9. Volba provozního módu
10. Ethernet připojení
11. Bezpečnostní usazení PLC
12. Port RS-232/RS-485-X3
13. Systémové LED indikátory
14. Tlačítko reset

1.1.3 Popis přípravku

PLC je připojeno na domácí ethernetové síti. Zprostředkovatelem je router ASUS WL-500G Premium s DHCP. K PLC je přistupováno buď skrze SSH nebo FTP server přes operační systém Ubuntu 14.04. K nahrávání vytvořených balíčků programů a rychlé nastavení je využíván Web Based Management defaultně dostupný skrze webový server Lighttpd.

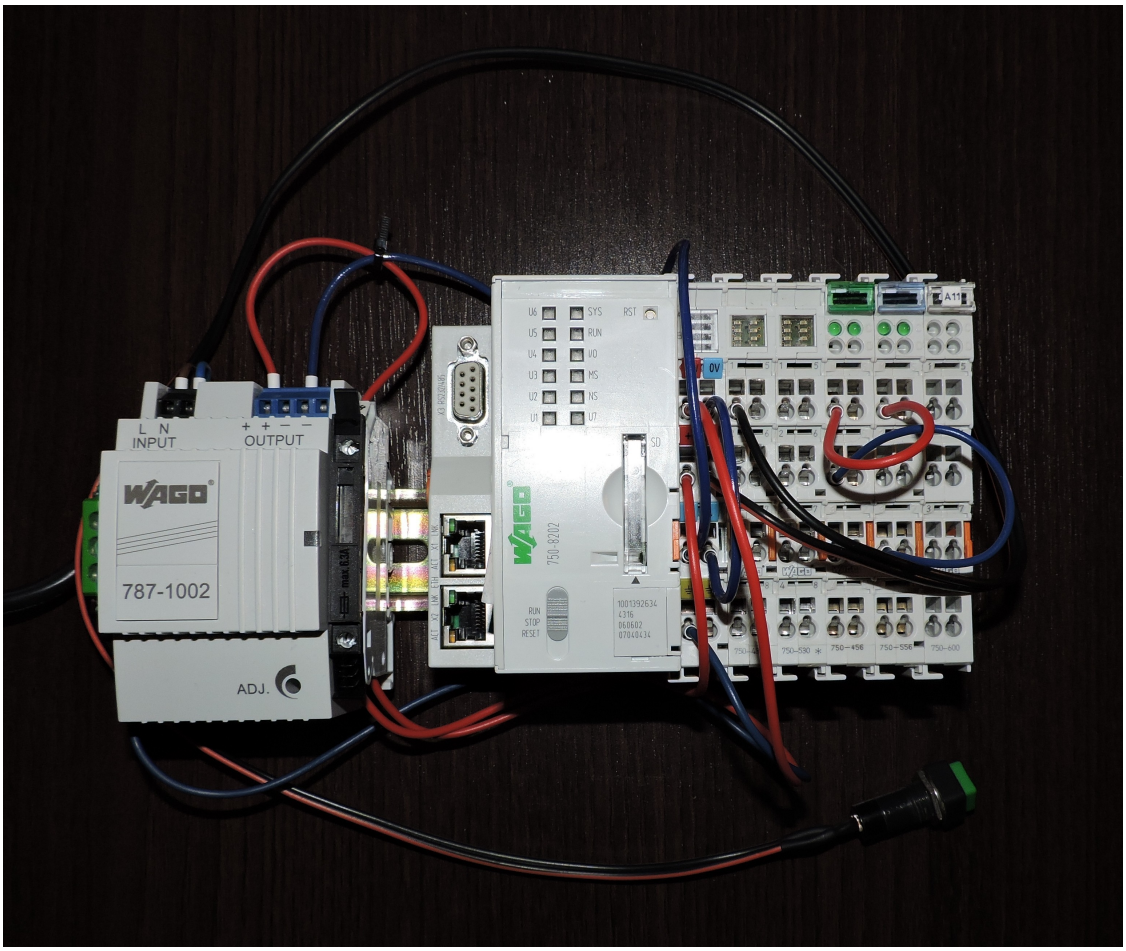


Obr. 1.3: Rozvržení domácí sítě

K PLC jsou připojeny tyto I/O moduly:

- 8 kanálový vstupní digitální modul 750-430 8 DI 24V DC
- 8 kanálový výstupní digitální modul 750-530 8 DO 24V DC
- 2 kanálový vstupní analogový modul 750-456 2 AI ±10V DC
- 2 kanálový výstupní analogový modul 750-556 2 AO ±10V DC
- Zakončovací modul 750-600

Na obrázku 1.4 můžeme vidět PLC PFC200 750-8202 s připojenými vstupními a výstupními moduly zmíněnými výše. Na prvním digitálním vstupu je připojené tlačítko a první analogový výstup je zapojen na první analogový vstup. Tato propojení jsou využívána pro pozdější testování v praktické části.



Obr. 1.4: Přípravek Wago PLC PFC200 s 24V zdrojem a I/O moduly

1.1.4 ADI/DAL

Hlavní předností PFC200, která jej odlišuje od ostatních, je jeho možnost dvojího způsobu ovládání. Jednou z nich je tradiční programování přes program CODESYS, případně e!COCKPIT, v jazycích daných normou IEC 61131-3. Druhou možností je přímé ovládání I/O modulů a případně paměti skrze skripty, které mohou být napsány v jiných programovacích jazycích, jakými jsou například C nebo C++. WAGO sám tuto iniciativu podporuje formou dodávaných "HowTos", ve kterých může programátor nalézt potřebnou inspiraci.

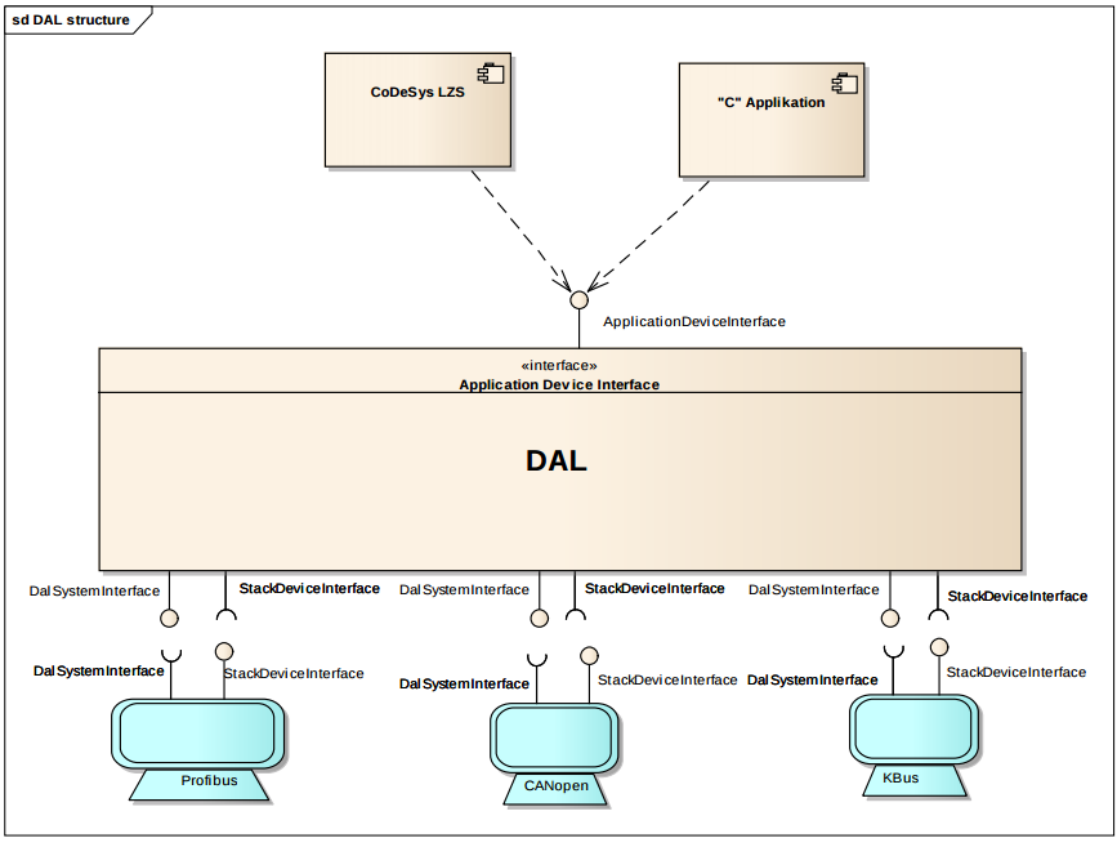
Pro komunikaci mezi aplikací, PLC a jednotlivými zařízeními na sběrnici je vytvořena DAL vrstva a rozhraní ADI, jejichž hlavním účelem je zjednodušení přístupu k jednotlivým komponentům a zavedení určitého standartu. Když neexistovala ADI/DAL, tak všechna zařízení komunikovala přímo s runtime běžící v PLC, což několikanásobně zvyšovalo požadavky na programátory a čas potřebný k vytvo-

ření dané aplikace. ADI/DAL tedy zastupuje roli administrátora mezi PLC, jeho zařízeními a jakoukoliv aplikací, která k nim požaduje přístup. Z toho vyplývá, že i CODESYS a e!COCKPIT musí využívat tuto vrstvu, jenže programátor komunikaci nevidí, pokud to zrovna nevyžaduje.

ADI/DAL aktuálně podporuje abstrakci a sjednocený přístup pro následující standardy, přičemž pro naši práci je využíván standard KBus:

Standardy podporované ADI/DAL

- PROFIBUS (Slave)
- CANopen
- ModbusTCP
- ModbusRTU
- KBus



Obr. 1.5: Využití ADI/DAL v PLC [2]

DAL - Device Abstraction Layer

Jedná se o ekvivalent HAL - Hardware Abstract Layer. Tato vrstva obsahuje všechny potřebné ovladače připojených zařízení. Transformuje požadavek zařízení na funkce obsažené v ADI vrstvě, se kterýma již dokáže CODESYS, e!COCKPIT nebo námi vytvořený skript komunikovat. To vše funguje i opačným směrem, nastane-li volání určité funkce, dojde k přeložení na instrukce podle daného ovladače pro zařízení.

ADI - Application Device Interface

ADI definuje potřebné typy funkcí, které musí každé zařízení dodat. Pro každý standard se jejich konkrétní provedení liší, avšak jejich účel zůstává stejný. ADI využívá proxy funkci "CallDeviceSpecificFunction", pomocí které si namapuje správnou funkci dle standardu. Jakákoliv aplikace, která tedy potřebuje komunikovat se zařízeními, musí obsahovat ADI/DAL hlavičku:

```
include <dal/adi_application_interface.h>
```

V této hlavičce jsou obsaženy základní funkce, které ADI nabízí pro komunikaci přes příslušný standard [2]:

Výpis 1.1: Výpis funkcí ADI

```
// System functions
int32_t (*Init)();
int32_t (*Exit)();
int32_t (*ScanDevices)();
int32_t (*GetDeviceList)();
int32_t (*OpenDevice)();
int32_t (*CloseDevice)();
bool (*IsDeviceOpen)();
int32_t (*GetDeviceName)();
int32_t (*GetDeviceFlags)();
int32_t (*GetIoSizes)();
// Configure devices(Out of scope of this document)
int32_t (*ConfigureSubdevice)();
int32_t (*ConfigureDevice)();
// Process data access
int32_t (*WriteStart)();
int32_t (*WriteBit)();
int32_t (*WriteBool)();
int32_t (*WriteBytes)();
```

```
int32_t (*WriteEnd)();
int32_t (*ReadStart)();
int32_t (*ReadBit)();
int32_t (*ReadBool)();
int32_t (*ReadBytes)();
int32_t (*ReadEnd)();
// Diagnostic
int32_t (*DiagnoseGetDeviceState)();
int32_t (*DiagnoseGetSubdeviceState)();
int32_t (*GetLastError)();
// Hooks
int32_t (*RegisterEventHandler)();
int32_t (*UnregisterEventHandler)();
// Devices operating mode
int32_t (*ApplicationStateChanged)();
// Proxy for device specific stuff
int32_t (*CallDeviceSpecificFunction)();
int32_t (*VCallDeviceSpecificFunction)();
// Watchdog, when device driven by external hardware/
   software
int32_t (*WatchdogSetTime)();
int32_t (*WatchdogStart)();
int32_t (*WatchdogStop)();
int32_t (*WatchdogTrigger)();
```

1.2 Pengutronix

Pengutronix je upravená distribuce Debianu vyvinuta v roce 2001 v Německu, specializovaná na embedded systémy v průmyslovém odvětví. Tento systém se vytváří pomocí nástroje PTXdist. Wago dodává ke svému PLC PFC200 sadu návodů, mezi kterými se nachází detailní průvodce instalací Board Support Package. Nutno podotknout, že tento návod je vytvořen pro vývojáře embedded linux systémů, je tedy potřeba mít dostupný Ubuntu 14.04, ať už na virtuálním disku nebo jeho úplnou instalaci. V Ubuntu 14.04 dojde k instalaci BSP a nástroje PTXdist, který vytváří kompletní image Pengutronixu s přednastavenými balíčky od firmy WAGO. V našem PLC se nachází nejaktuálnější verze BSP z března 2017 (WAGO-PFC-BSP-2017.3.1).

PFC200 má dvojí možnost bootování systému. Na vnitřní flash paměti je ulo-

žena čistá instalace Pengutronixu. Pro usnadnění vývoje je možnost do PLC zapojit i externí SD kartu, na kterou můžeme nahrát námi vytvořený image Pengutronixu, klidně i novější verze. PLC vždy dává přednost SD kartě. Pokud je na ní nahrán poškozený nebo neúplný image, vrátí se zpět do své interní flash paměti.

Jakmile v PLC existuje funkční verze Pengutronixu, může k němu uživatel přistupovat pomocí SSH, případně Web Based Managementu (WBM) běžícím na webovém serveru. Defaultně jsou vytvořeny profily root, admin a user. Pro důležité systémové úpravy je potřeba využívat práva root-u. Přes WBM si může uživatel nastavit další možnosti komunikace jako je FTP, HTTP apod.

1.2.1 PTXdist

Politika Pengutronixu je nastavená tak, že přímé binární distribuce jejich operačního systému jsou příliš neflexibilní pro rychle se vyvíjející embedded systémy. Je proto lepší dodávat nástroj PTXdist, který slouží ke kompilaci cílového systému přímo z originálních zdrojových kódů. Tato kompilace je řešena jako sled potřebných kroků k vytvoření cílového systému, které může zkušený uživatel měnit podle svého a ten méně zkušený nemusí zasahovat vůbec. PTXdist využívá OSELAS.Toolchain(), který dodává potřebné nástroje pro cross-kompilaci embedded Linux projektů: GCC, binutils, gdb a glibc. Cross-kompilaci z toho důvodu, že v našem PLC je procesor pracující na architektuře ARM.[3]

Jakmile nastane situace, kdy potřebujeme do PLC nahrát námi vytvořený program, napsaný například v jazyce C/C++, je nutno z jeho zdrojových souborů vytvořit balíček .ipk pomocí nástroje PTXdist a posléze jej nahrát do PLC. K tomu slouží následující příkaz:

```
ptxdist newpackage <type>
```

Nejčastěji budeme pracovat s typy balíčku *target* nebo *file*. *Target* stáhne danou verzi aplikace přímo od výrobce (takto jsou vytvořeny již přednastavené balíčky od firmy Wago). *File* využije již existující zdrojové soubory. Ať už vybereme jakoukoliv verzi balíčku, musíme vyplnit potřebné informace:

```
ptxdist: enter package name.....: XXX
ptxdist: enter version number.....: 1.1.0
ptxdist: enter URL of basedir.....: http://www.XXX.com/download/src
ptxdist: enter suffix.....: tar.gz
ptxdist: enter package author.....: My Name
ptxdist: enter package section....: project\_specific
```

Dojde k vytvoření rules (pravidel) ve složce Rules, vytvoří se soubor XXX.in a XXX.make, které v sobě mají uložené potřebné informace k vytvoření balíčku. Tyto soubory se dají upravovat - můžeme například přidat dodatečné knihovny apod. Abychom balíček přiřadili do naší distribuce, musíme jej povolit v nastavení PTXdist, do nejž se dostaneme příkazem:

```
ptxdist menuconfig
```

Poté sekvencí následujících příkazů vytvoříme .ipk balíček:

```
ptxdist get XXX
ptxdist extract XXX
ptxdist prepare XXX
ptxdist compile XXX
ptxdist install XXX
ptxdist targetinstall XXX
```

Jakmile dojde k úspěšnému vytvoření balíčku XXX.ipk, zbývá pouze jeho nahrání do systému. To můžeme udělat ručně nebo pomocí Web Based Managementu. Kdyby se náhodou jednalo o rozsáhlý balíček, doporučuje se vytvořit nový image Pengutronixu. [3]

Nutno v této části podotknout, že pokud náš program přistupuje k ADI/DAL vrstvě, je potřeba vypnout proces plclinux_rt, jelikož k ADI/DAL může najednou přistupovat pouze jeden proces. Defaultně k němu přistupuje plclinux_rt, případně e!runtime.

1.2.2 PLC runtime systém

Nejdříve je nutno vysvětlit, co to vlastně runtime je. Konkrétně runtime system, do češtiny volně přeloženo jako "běhové prostředí". Jedná se o soubor softwarových a hardwarových prostředků, které umožňují externímu programu, aby byl spuštěn na daném cílovém systému. Typicky obsahuje nízkoúrovňové příkazy, které dokáží přistupovat k přítomnému hardwaru - instrukce procesoru, práce s pamětí, konverze do binární soustavy apod. Také však obsahuje příkazy vysokoúrovňové, které dokáží pracovat s primarním softwarovým frameworkem nebo knihovnami.[7]

V PFC200 se nachází defaultně dva druhy runtime, určené pro svá vývojová prostředí. Můžeme si však vytvořit i svůj vlastní, nezávislý na těch defaultních, pomocí daemonu, který běží na pozadí a umožňuje nám komunikovat s námi vybranými zařízeními. Ve Web Based Managementu je možné tento runtime změnit.

CODESYS2 runtime systém

Runtime, který využívá vývojové prostředí CODESYS. Naalokovává si 16 MB paměti pro program, 64 MB paměti pro data a 128 KB jakožto NVRAM.

e!RUNTIME systém

Runtime, který je určen pro e!COCKPIT. Naalokovává si celkově 60MB paměti, kterou dynamicky rozděluje pro program a data, a také 128 KB jakožto NVRAM.

1.2.3 Daemon runtime systém

Může nastat situace, kdy programátor nevyužije původní runtime systémy a potřebuje si vytvořit svůj vlastní. V první řadě musíme vypnout defaultně nastavený runtime. Posléze je potřeba vytvořit program, který bude neustále v cyklu probíhat na pozadí ihned po nabootování do systému a obsluhovat požadavky uživatele. K tomuto se skvěle hodí vytvořit tzv. daemon program.

Daemon není nic jiného než program, který se však nenachází v přímém kontaktu s uživatelem. Jeho smyslem je vyčkávat na nějakou událost, kterou mu programátor stanoví, tu obsloužit a tím tedy zajišťuje potřebné úkony bez nutnosti komunikace s uživatelem. Pokud jej správně nastavíme, tak jej můžeme využít jako runtime systém, který v našem případě bude obsluhovat přesně daná zařízení připojená k PLC

s využitím ADI/DAL vrstvy.[8]

Vytvoření daemon programu

Vytvoření daemon programu má pro každý operační systém jiný postup. My se budeme věnovat pouze tomu pro UNIX. Nejdříve je důležité položit otázku, jaký úkol má daemon plnit. Pro náš případ má tedy daemon zastávat funkci runtime systému a umožňovat obsluhu HW komponentů. Poté již následuje sled základních kroků pro vytvoření daemona:

Kroky k vytvoření UNIX daemona[8]

- Rozdělení tzv. parent procesu
  - Daemon je spuštěn systémem nebo uživatelem. Jakmile dojde k jeho spuštění, je daemon jako každý jiný program v systému. Aby se stal autonomním, musí se vytvořit tzv. child proces, ve kterém se spouští samotný kód. Tomu se říká dělení - forking. Využívá se fork() funkce.
- Změna masky souborového módu (umask)
  - Abychom mohli zapisovat/číst z jakýchkoliv souborů (například log) vytvořených daemonem, je potřeba nastavit umask na 0. Díky tomu budeme mít plný přístup k souborům generovaným daemonem.
- Otevření log souborů pro zápis
- Vytvoření unikátní SID (Session ID)
  - Child proces potřebuje svůj unikádní Session ID aby mohl nerušeně operovat. Jinak se z něj stane tzv. orphan proces (sirotek).
- Změna aktuálního pracovního adresáře na bezpečný adresář
  - Aktuální pracovní adresář se změní na takový, jehož existence bude předem zajištěna. Defaultně se tento adresář nastavuje na root (/).
- Uzavření otevřených souborů (deskriptorů)
  - Jelikož daemon nemůže využívat terminál, jsou tyto deskriptory (STDIN, SDTOUT, STDERR) zbytečné a potenciálně nebezpečné. Deskriptor je abstraktní ukazatel, který se používá pro přístup k souboru nebo jiného vstupního/výstupního prostředku (např. socketu).
- Vložení samotného daemon kódu
  - Zde je prostor pro náš kód, který se bude cyklicky opakovat (while cyklus) a podle nastavení uživatele bude umožňovat operace na pozadí systému.

1.3 Webové nástroje

Abychom docílili touženého výsledku dynamických webových stránek, které dokáží reagovat v reálném čase na podněty uživatele bez nutnosti obnovení celých stránek, je potřeba skloubit několik technologií dohromady. Idea je taková, že pomocí HTML navrhneme základní rozvržení stránek, JavaScript zajistí dynamickou reakci na podněty uživatele a PHP využijeme jako skript pro zápisy do souboru.

1.3.1 HTML

HTML, neboli HyperText Markup Language je jeden z nejjednoduších jazyků umožňujících tvorbu webových stránek. HyperText je metoda, pomocí které se můžeme pohybovat na webu - klikáním na speciální odkazy zvanými hyperlinky, které nás přemisťují na jiné stránky. Hyper znamená, že neexistuje žádná linearita, mohu se dostat kamkoliv odkudkoliv. Markup označuje funkci HTML tagů. Jazyk HTML je spravován neustále rostoucí online komunitou W3C, World Wide Web Consortium.

Jak vlastně HTML funguje? Jako každý jiný jazyk má své typické výrazy a syntax. Skládá se ze série kratších kódů zapsaných do tzv. tagů, které uživatel zapisuje do .html souboru. Internetové prohlížeče typu Google Chrome, Internet Explorer nebo Mozilla Firefox tyto soubory dokáží číst a přeložit text na viditelnou formu. Tagy oddělují běžný text od HTML kódu a každý z nich plní různou funkci, například formátování písma nebo rozložení textu.[9]

HTML podporuje vytváření CSS nebo využití jiných jazyků jako například Javascript nebo PHP. CSS znamená Cascading Style Sheets. Tato CSS popisují, jak se mají elementy zobrazovat na našich stránkách a usnadňují programátorům práci, jelikož fungují podobně jako funkce a dají se využít na jakýchkoliv stránkách.[9]

O pár řádků níže následuje příklad nejjednoduších HTML stránek. Tag <head> značí hlavičku naší stránky a <body> pak samotný obsah.

Výpis 1.2: Příklad HTML kódu

```
<html>
<head>
    <title>My test page</title>
</head>
<body>

   Text

</body>
</html>
```

1.3.2 PHP

PHP - Hypertext Preprocessor je open source skriptovací jazyk, který dokáže být součástí HTML kódu. Pokud potřebujeme nějaký skript s funkcí, kterou HTML nedokáže zvládnout, můžeme si napsat tyto skripty v jazyku jako je například C nebo Perl. Jejich volání je však složité a HTML stránky vyžadují výstupní HTML příkazy z těchto skriptů. Oproti tomu nám PHP umožňuje přímé volání těchto skriptů v HTML kódu.[10]

Výpis 1.3: Příklad PHP kódu

```
<?php
            echo "This is my simple PHP script!";\\
        ?>
```

PHP kód je spouštěn na serveru a generuje HTML, které je posléze posíláno klientovi. Klient obdrží výsledky daného skriptu, avšak nedokáže určit přesný kód, který tento skript vykonal. Jedná se tedy o jeden z možných způsobů vykonávání dynamického obsahu.

1.3.3 JavaScript

Pro naši práci je JavaScript stěžejní nástroj. Jedná se o programovací jazyk, který umožňuje vytvoření dynamicky aktualizovaného obsahu stránek, kontrolu multimédií, animace a podobně. Oproti PHP, který je server-side je Javascript client-side. To znamená, že samotný JavaScriptový kód je stažen k uživateli, spuštěn a posléze zobrazen webovým prohlížečem.[11]

Jeho syntaxe je podobná ostatním programovacím jazykům jako například C. V JavaScriptu si můžeme deklarovat proměnné různých datových typů jako je například integer, float a jiné. Respektive jakmile dojde k vytvoření proměnné příkazem *var*, JavaScript ji sám přiřadí datový typ podle toho, co do této proměnné ukládáme. S těmito proměnnými pak můžeme pracovat pomocí metod a reagovat na podněty uživatele pomocí eventu. Takovým typickým eventem je například *.change()*, reagující na změnu hodnoty v ovládacím prvku. Kromě toho můžeme využívat známé programátorské metody jako cykly (for, while), podmínky (if) nebo funkce.

Jak tedy vlastně Javascript na webu funguje? Javascript je spuštěn v javascript enginu daného webového prohlížeče. Následně HTML, případně s pomocí CSS, vytvoří defaultní rozložení stránky. Javascript posléze dynamicky modifikuje HTML (CSS) aby aktualizoval webovou stránku. Kód se vykonává v pořadí, v jakém je zapsán - jednoduše řečeno od shora dolů.[11]

Výpis 1.4: Příklad JavaScript kódu

```
<script>

var number = 0;      //deklarace proměnné

function increment() {        //funkce
    number = number + 1;
    }

\$("#button").change(function(){      //event
    increment();          //volání funkce
   });

</script>
```

JSON soubory

JSON je zkrácená verze JavaScript Object Notation. JSON nám poskytuje nový způsob ukládání informací/dat v organizované a jednoduše přístupné struktuře. Souhrnně, JSON soubory umožňují přehledný zápis dat, ke kterému můžeme přistupovat jednoduchou logikou. Jedná se o obdobu XML souborů, která je využívaná primárně v jazyce JavaScriptu. Existují v něm totiž funkce, které umožňují rychlé rozdělení dat obsažených v JSONu do proměnných v našem skriptu, čímž se programátorovi usnadní spousta práce, jelikož nebude potřeba nastavování čtení dat ze souboru a

dalšího kódu.

Výpis 1.5: Příklad syntaxe JSON souborulanguage

```
{
    "an_read1":72,
    "an_read2":88,
    "dig_read":[0,0,0,0,0,0,0,0]
}
```

JavaSript moduly - JQuery

JavaScript dokáže podobně jako jiné programovací jazyky využívat externí knihovny, které usnadňují programátorům práci. Pro přidání této knihovny stačí jednoduchý příkaz podobný **#include** :

```
<script src="jquery-3.2.1.min.js"></script>
```

JQuery je využíván i v této práci. Jedná se o javascriptovou knihovnu podporující většinu známých webových prohlížečů. Rozšiřuje javascript o další funkce a metody, jako například AJAX, efekty a animace, procházení a změna DOM apod.

1.3.4 AJAX

AJAX - Asynchronous JavaScript and XML je nástroj pro tvorbu rychlejších a interaktivnějších webových aplikací s pomocí XML, HTML, CSS a JavaScriptu. Jedná se o technologii webových prohlížečů, která je nezávislá na softwaru webového serveru.

Na běžných stránkách, které nevyužívají tuto metodu, uživatel něco vykoná, vyšle požadavek webovému serveru a je přesměrován na novou stránku s novými informacemi. Při využití AJAX však uživatel ani nepozná, že došlo k vyslání požadavku. JavaScript vyšle požadavek na server, interpretuje výsledky a upravuje aktuální otevřenou stránku - a to vše probíhá na pozadí bez vědomí uživatele.

AJAX nedokáže pracovat nezávisle. Využívá se v kombinaci s dalšími technologiemi. Základem je JavaScript, DOM (rozhraní pro přístup a manipulaci dokumentů), CSS a XMLHttpRequest (objekt javascriptu, který provádí asynchronní interakci s webovým serverem). [14]

1.3.5 Způsob spouštění server-side skriptů - CGI, FastCGI

Abychom dokázali realizovat spouštění skriptů na straně serveru, které budou vykonávat například funkci zápisu nebo čtení ze souboru, je nutné, aby existoval jistý druh mediátoru, který dokáže webovému serveru zprostředkovat spuštění těchto skriptů. Pro případ této práce se jedná o využití rozhraní/protokolu CGI nebo FastCGI.

CGI

Common Gateway Interface nabízí webovým serverům standardizovaný protokol, který popisuje a realizuje komunikaci mezi programem a webovým serverem. CGI byl v minulosti jediný způsob, jak vytvářet webové stránky s dynamickým obsahem. Dnes se již však vytlačuje moduly skriptovacích jazyků, jako třeba PHP, Python atd. Způsob fungování CGI je následující: Program získává data a informace od serveru z různých vstupů. Zpět tomuto serveru a posléze klientovi posílá CGI program odpověď standardním výstupem.[12]

FastCGI

Jedná se o vylepšenou verzi CGI. Původní CGI při každém požadavku načítá program znovu a spouští jej. FastCGI však nabízí jiný přístup - program se spustí pouze při prvním požadavku a běží ve smyčce, ve které obsluhuje všechny budoucí požadavky.[13]

1.4 Rešerše webových serverů podporující Linux

V následující kapitole se budeme věnovat výběru vhodného webového serveru pro náš systém. Jelikož se jedná o embedded systém s omezenými hardwarovými zdroji, je nutno dbát převážně na potenciální zatížení. Následně je vhodné se zaměřit na technologie podporující dynamické webové rozhraní, jakými je například CGI.

1.4.1 Apache HTTP server

Apache je nejpoužívanější a jeden z nejstarších webových serverů, který je dostupný veřejnosti již od roku 1995. V dnešní době se využívá zhruba na 60% všech webových stránek, ale jeho využití klesá, hlavně díky nízkému výkonu a stabilitě. Tento webový

server podporuje operační systémy Windows, Mac OS X, Unix a další. Apache umožňuje využití běžných programovacích jazyků jako Perl, Python, Tcl nebo PHP. Pro přidání dalších funkcí slouží přídavné balíčky, které se instalují k jádru serveru. [4]

Díky tomu, že je již na trhu dostupný dlouhou dobu, existuje mnoho distribucí a balíčků, obsahující samotný Apache HTTP server společně s jinými webovými aplikacemi, jako například databáze MySQL, PHP a s podporou pro různé operační systémy. [4]

1.4.2 Nginx

Nginx je druhý nejvyužívanější webový server, ihned po Apache. Stává se pomalu ale jistě jeho nástupcem hlavně díky své jednoduchosti, lepšímu využití paměti a celkově vyššímu výkonu. Může se také využít jako IMAP/POP3 reverzní proxy server. Podporuje operační systémy jako Unix, Mac OS X, Windows a další. [4]

Při vývoji Nginxu se kladl důraz na výkon. Konkrétně aby dokázal zvládnout tisíce připojených klientů najednou. Cílem je tedy rychlá distribuce statického obsahu a možnost rozložení zátěže na další servery dle priority (využití jako reverzní proxy). Příchozí požadavky Nginx zpracovává a vyřizuje asynchroně, narozdíl od Apache, který využívá vlákna nebo procesy. Pro představu, pokud dojde od klienta požadavek, v ten moment je kontrolována cache paměť pro rychlou odpověď. Pokud nenajde odpověď na požadavek v této cache paměti, podle priority vysílá požadavek dále na některý ze serverů, který zpětně vyšle odpověď nebo se požadavek posílá dále. Podobně jako u Apache lze do Nginxu instalovat moduly, které například umožňují dodatečné zabezpečení, streamování, přesměrování a tak dále. [4]

1.4.3 Lighttpd

Lighttpd je oblíbený pro svou jednoduchost a nenáročnost. Přesně tyto vlastnosti z něj dělají ideálního kandidáta pro embedded systémy, jako například Raspberry Pi nebo také naše PLC PFC200. Pracuje podobně jako Nginx, asynchronně zpracovává požadavky od uživatelů. [5]

Mezi jeho hlavní přednosti patří efektivní využití paměti, FastCGI, SCGI, PHP, WebDNA (skriptovací jazyk s vestavěným databázovým systémem) a spoustu dalších. [5]

1.4.4 Cherokee

Cherokee je nenáročný a hlavně přístupný pro nezkušené uživatele díky svému jednoduchému konfiguračnímu GUI. Podobně jako Lighttpd, podporuje velké množství moderních aplikací: FastCGI, SCGI, PHP, CGI, SSI, TLS/SSL, Django a tak dále. Jedinou a celkem značnou nevýhodou je chybějící podpora od vývojářů. Poslední update, který pro tento server vyšel, je z roku 2011, což je hlavně z důvodu bezpečnosti velice nevhodné. [5]

1.4.5 Monkey HTTP Daemon

Monkey HTTP Daemon je další webový server, který je podobně jako Lighttpd mířen převážně na embedded systémy. Jedná se taktéž o asynchronní webový server. Funguje na bázi pluginů, podporuje CGI, SSL, dodatečnou bezpečnost, logování a podobně. Monkey může být využít jako stand-alone server pro statické stránky nebo díky skriptovacím jazykům jako PHP, Perl, Python nebo Lua může poskytnout FastCGI rozhraní pro dynamické webové stránky. Jedná se tedy o skvělou alternativu Lighttpd. [5]

1.4.6 Hiawatha

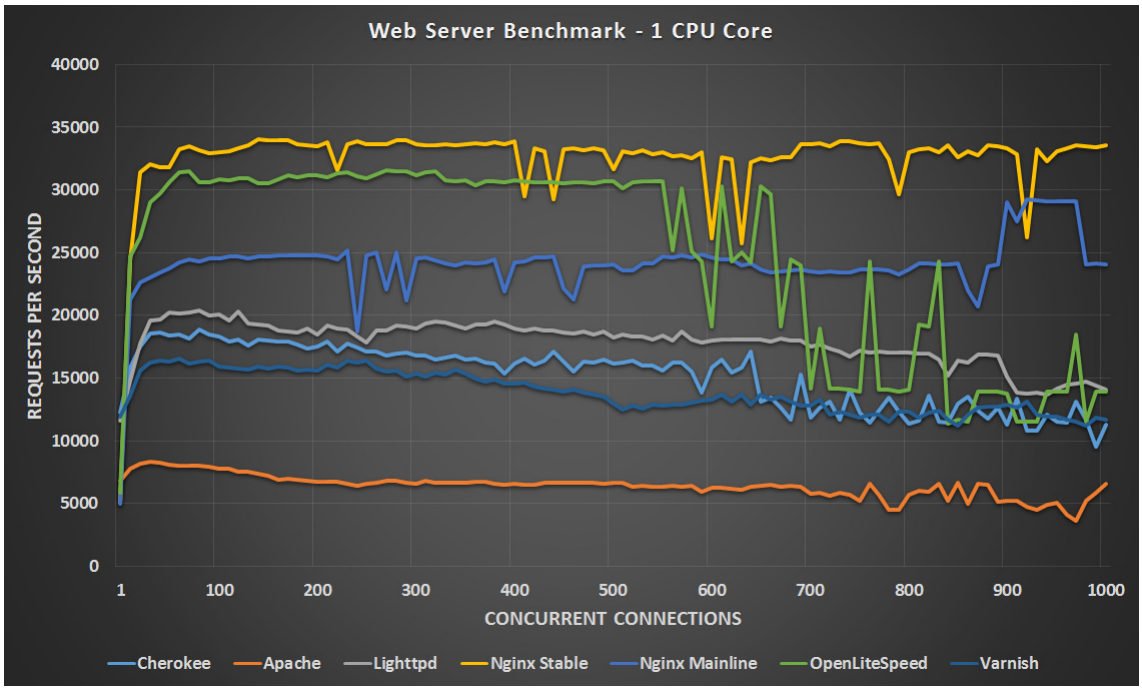
Hiawatha je webový server orientovaný převážně na bezpečnost a jednoduchost obsluhy. Hodí se skvěle k vytváření dynamických stránek. Mezi jeho přednosti patří podpora přenosu velkých souborů, reverzní proxy, SSL/TLS, FastCGI a dále. [5]

Oproti ostatním serverům vyniká svými implementovanými bezpečnostními opatřeními. Dokáže zastavit nechtěné zásahy do databází SQL, banovat potenciální hackery, omezuje runtime CGI aplikací, exploitů a spoustu dalšího. Obsahuje také vestavěný monitorovací nástroj pro kontrolu všech webových serverů. [5]

1.4.7 Porovnání výkonu webových serverů

Webové stránky rootusers.com provedly rozsáhlý test nejvyužívanějších webových serverů, které podporují operační systém Linux. Testován byl hlavně průběh výkonu

v závislosti na počtu připojení a požadavků. Dále také využití paměti a rychlost zpracování benchmarku. Test byl proveden pro různý počet jader procesoru. Pro nás je důležitý hlavně výsledek pro jednojádrové procesory, jelikož v našem PLC PFC200 je přítomný jednojádrový procesor ARM Cortex A8.



Obr. 1.6: Porovnání výkonů webových serverů [6]

Jak můžeme z obrázku 1.6 vyčíst, Apache nabízí stabilní, avšak nízký výkon, který při vyšším počtu připojených uživatelů začíná kolísat. Nginx nabízí nejvyšší výkon s občasnými výkyvy, avšak tento výkon by byl v našem případě nevyužitelný. Cherokee má minimální výkonnostní rozdíl oproti Lighttpd, avšak při vyšších připojeních opět začíná výrazně kolísat. Lighttpd je dobrým kompromisem mezi těmito servery, nabízí stabilní a přitom dostatečný výkon i při vyšším počtu připojení. Ostatní webové servery vyobrazeny v grafu nejsou brány v potaz, jelikož se jedná o ne příliš využívané a vhodné webové servery pro náš účel. Co se týče využití paměti, jejíž benchmark je vyobrazen na obrázku 1.7, Apache zaostává se skoro trojnásobně vyšším využití paměti oproti Nginx, Lighttpd nebo Cherokee. V poměru výkon/náročnost nám tedy z těchto grafů vychází nejlépe Lighttpd.

Obr. 1.7: Porovnání využívání paměti webových serverů [6]

1.4.8 Výběr webového serveru

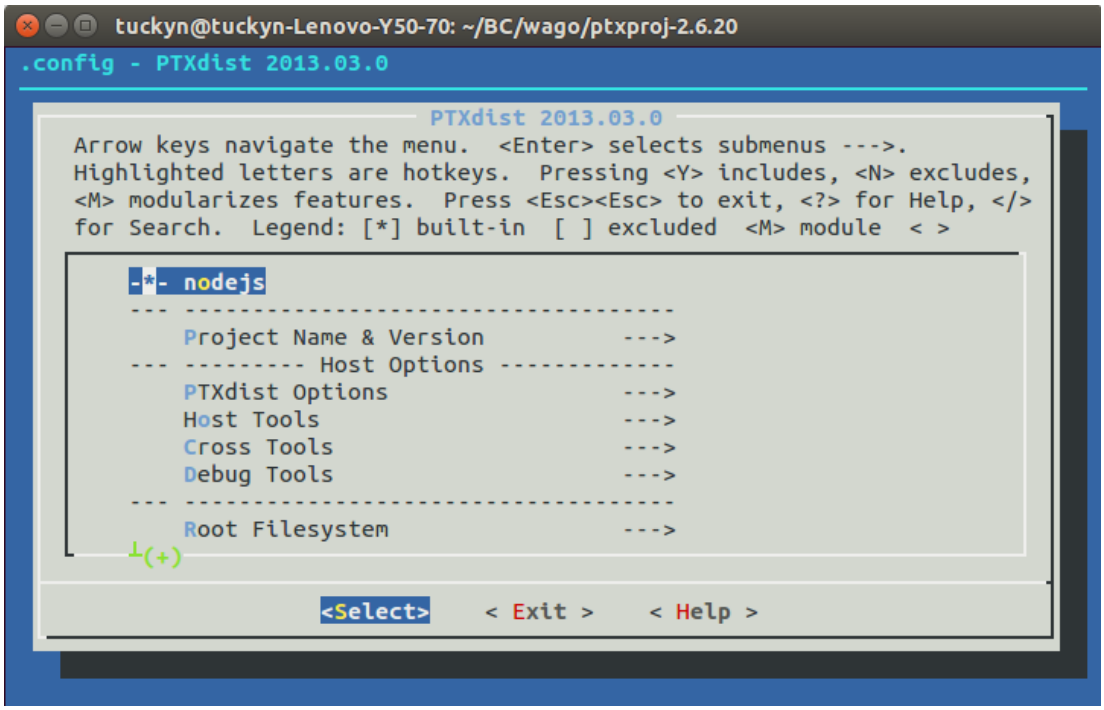
Jaký webový server je tedy nejvhodnější? Z výše uvedených připadají v úvahu Lighttpd nebo Monkey HTTP Daemon. Hiawatha je taktéž vhodný kandidát, avšak pro naši aplikaci je zvýšená bezpečnost nevyužitelná. Lighttpd nebo Monkey jsou webové servery přímo vytvořené pro embedded systémy, vzhledem k tomu, že tyto systémy nedisponují příliš výkonným hardwarem.

V potaz je brán ten fakt, že na PFC200 je již defaultně přítomný webový server Lighttpd. Společnost WAGO vydává distribuci Pengutronixu, ve které se již nachází předinstalované programové balíčky, které jsou odzkoušené a vyladěné pro náš systém. Jedním z těchto balíčků je právě webový server Lighttpd. Při http požadavku přes port 80 je uživatel defaultně přesměrován na stránky tzv. Web Based Managementu, který umožňuje měnit základní nastavení PLC, jako je například volba runtime systému, nahrávání dalších programových balíčků nebo povolení různých komunikačních protokolů. Pokud bych se tedy rozhodl k instalaci jiného webového serveru, mohlo by docházet k nechtěným konfliktům mezi těmito aplikacemi. Nevidím tedy důvod v nucené instalaci jiného webového serveru, když už na PLC jeden existuje a je vyladěn pro naše použití. Navíc bez Web Based Managementu bychom zbytečně obtížně nastavovali základní konfiguraci našeho PLC.

2 PRAKTICKÁ ČÁST BAKALÁŘSKÉ PRÁCE

2.1 Instalace Pengutronixu

Abychom mohli využívat výhody embedded Linux prostředí PFC200, je potřeba získat upravenou distribuci Unixu - Pengutronix, která již obsahuje předinstalované programové balíčky s aplikacemi, které může programátor využít. K tomu je potřeba stažení BSP - Board Support Package, vytvoření obrazu distribuce Pengutronixu a jeho nahrání na SD kartu v PLC. Wago ke svému PLC PFC200 dodává dokumentaci, ve které detailně popisuje kroky instalace BSP a vytvoření přídavných programových balíčků. Návod je určen pro vývojáře pracující v Linuxovém prostředí, bylo proto nutné nainstalovat Ubuntu verzi 14.04. Instalace proběhla na vyčleněný prostor na disku.



Obr. 2.1: PTXdist menuconfig

Základem je buildovací nástroj PTXdist, který dokáže vytvořit image naší distribuce Pengutronixu a také vytvářet nové programové balíčky. Po instalaci všech potřebných programů a knihoven může uživatel pomocí příkazu *ptxdist menuconfig* nastavit konkrétní konfiguraci vytvářeného image - například přidat ukázkové programy od WAGA, různá rozšíření bezpečnosti, Node.js apod.

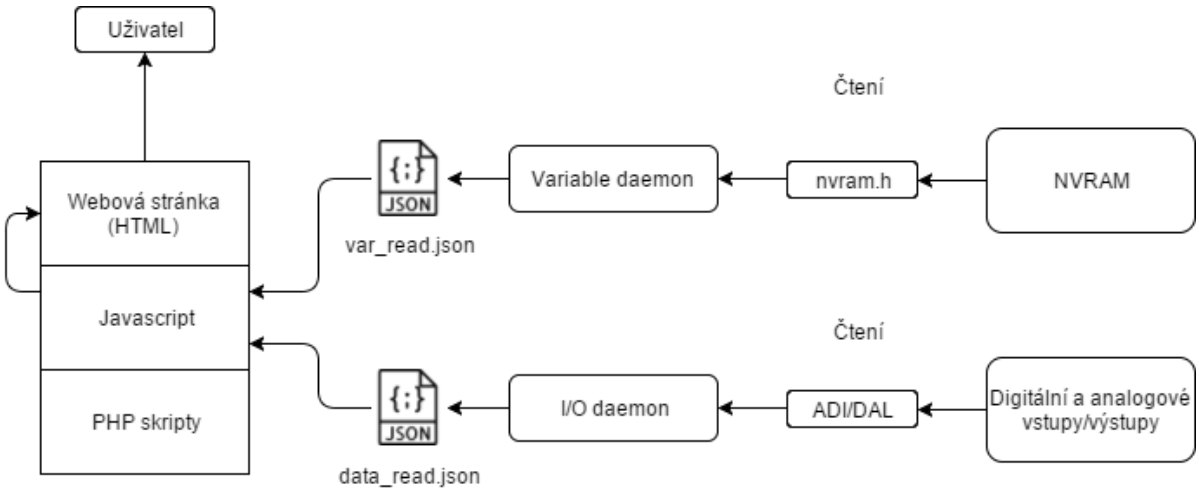
Pomocí následujících příkazů pak dojde k vytvoření samotného image sd.hdim.

```
ptxdist go -q
ptxdist images
```

Poté již stačí zapsat vytvořený soubor sd.hdim na kartu SD a tu vložit do PLC.

2.2 Popis řešení zadání

Po instalaci Pengutronixu na SD kartu máme vytvořeno prostředí, ve kterém můžeme realizovat cíle bakalářské práce. Pro řešení zadání jsem zvolil cestu čtení/zapisování skrze JSON soubory, ke kterým zároveň přistupují jak webové stránky, tak námi vytvořený daemon program.

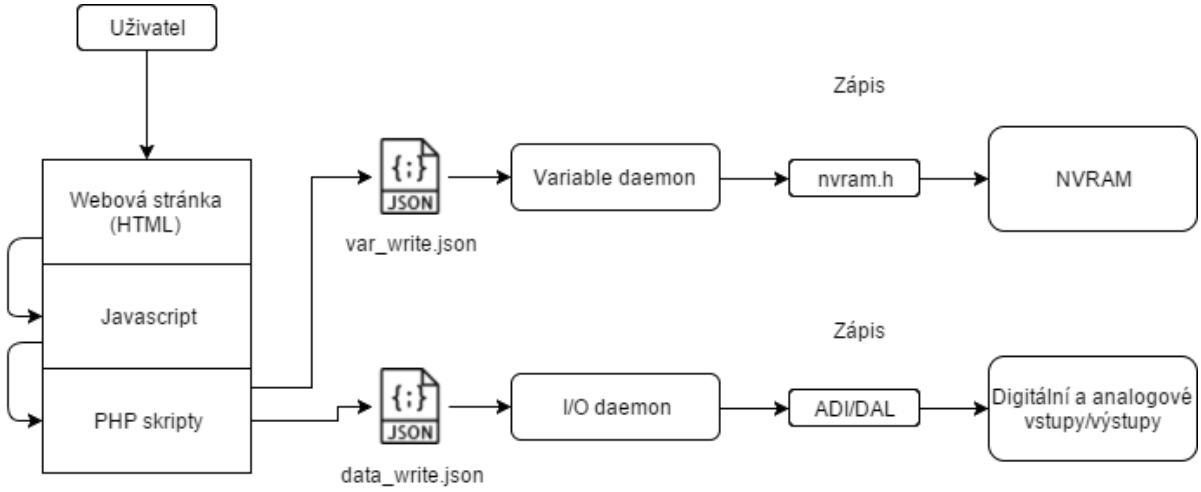


Obr. 2.2: Diagram řešení čtecích úkonů

Čtení dat, ať už proměnných z paměti nebo vstupů/výstupů, probíhá cyklicky. Variable a I/O daemon co 10ms aktualizují hodnoty zapsané ve speciálním formátu v souboru *var_read.json* a *data_read.json*.

Variable daemon využívá funkce obsažené v knihovně nvram.h, které inicializují NVRAM paměť a provádějí s ní potřebné úkony - v tomto případě čtení. Naopak I/O daemon využívá vrstvy ADI/DAL a jejich funkcí ke čtení hodnot na vstupních modulech PLC. Dvoukanálového analogového a digitálního s osmi vstupy.

Zápis hodnot je již o něco složitější, jelikož webová aplikace neumožňuje přímý zápis do souboru, jak je tomu u čtení. Je tedy nutno vytvořit skript, který se bude



Obr. 2.3: Diagram řešení zapisovacích úkonů

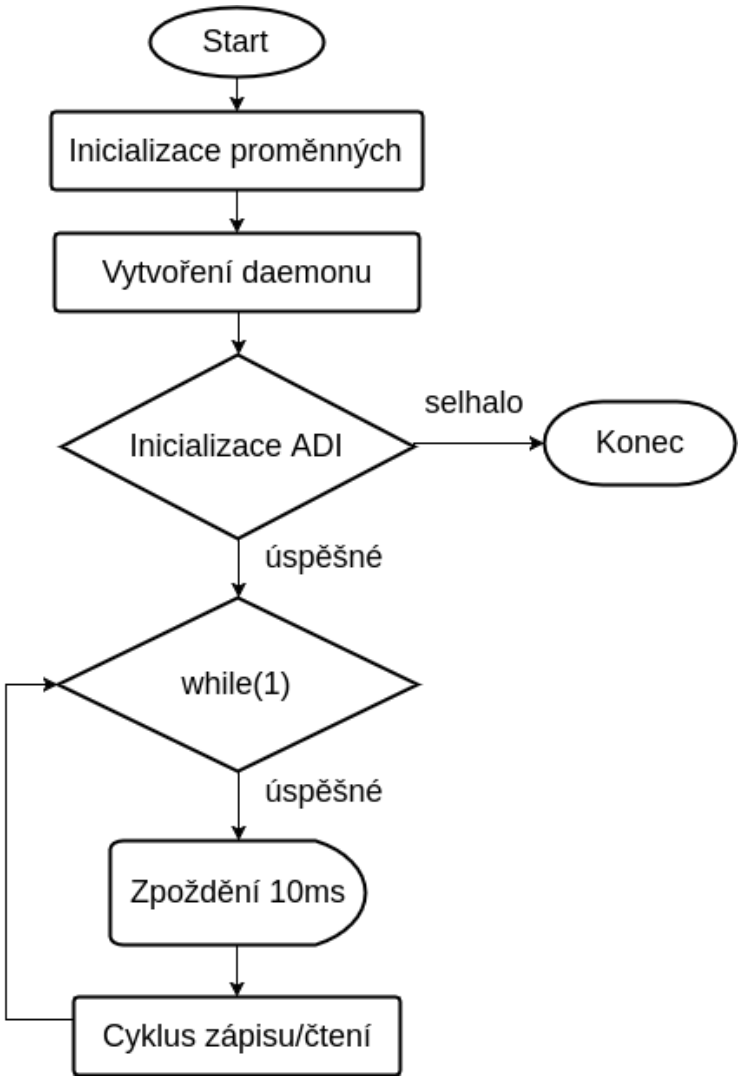
spouštět na straně serveru, který nám tento zápis bude zprostředkovávat. Jsou vytvořeny dva PHP skripty, které jsou volány skrze Javascriptový příkaz GET, jsou jim předána data a ty jsou ukládány do specifických souborů. Podobně jako u čtení Variable daemon využívá k zápisu knihovnu nvram.h a I/O daemon funkce vrstvy ADI/DAL.

2.3 Daemon programy

Máme tedy připravené PLC a promyšlené řešení našeho zadání. Následuje vytvoření daemon programů, které budou zastávat funkci runtime systému a umožňovat uživateli komunikovat s hardwarovými komponenty. V následující sekci si popíšeme důležité funkce našich programů v takovém pořadí, v jakém následují (podle diagramů). Záměrně zde nejsou vypsány celé kódy, jelikož by docházelo ke zbytečnému prodlužování bakalářské práce a autor předpokládá základní technické znalosti čtenáře v oblasti programování.

2.3.1 I/O daemon

Následující diagram popisuje základní funkci daemona, zabezpečujícího zápis a čtení vstupních/výstupních modulů připojených k PLC skrze standard KBus. Část výsledného kódu vychází z ukázkových programů přímo od firmy WAGO. Způsob využití ADI/DAL vrstvy ke komunikaci se zařízeními na KBus je jednotný a proto existuje velmi malá možnost úpravy kódu, abychom docílili správné funkčnosti.



Obr. 2.4: Diagram daemon programu pro zápis/čtení vstupů a výstupů PLC

Abychom mohli využívat funkce, které nám ADI nabízí, je nutno zahrnout do našeho programu ADI knihovnu:

```
#include <dal/adi_application_interface.h>
```

Výpis 2.1: Inicializace ADI

```
adi = adi_GetApplicationInterface();
adi->Init();
adi->ScanDevices();
adi->GetDeviceList(sizeof(deviceList), deviceList, &
    nrDevicesFound);

nrKbusFound = -1;
for (i = 0; i < nrDevicesFound; ++i)
{
  if (strcmp(deviceList[i].DeviceName, "libpackbus") == 0)
  {
    nrKbusFound = i;
  }
}
if (nrKbusFound == -1)
{
  adi->Exit();
  return -1;
}
s_param.sched_priority = KBUS_MAINPRIO;
sched_setscheduler(0, SCHED_FIFO, &s_param);

kbusDeviceId = deviceList[nrKbusFound].DeviceId;
if (adi->OpenDevice(kbusDeviceId) != DAL_SUCCESS)
{
  adi->Exit();
  return -2;
}
  event.State = ApplicationState_Unconfigured;

  if (adi->ApplicationStateChanged(event) != DAL_SUCCESS)
  {
  adi->CloseDevice(kbusDeviceId);
    adi->Exit();
    return -3;
  }
```

Tento kód popisuje nutnou inicializaci a vyhledávání zařízení komunikujících skrze KBus. V teoretickém rozboru jsme si popsali základní funkce ADI, které musí každý komunikační standard podporovat.

V deklaraci proměnných dochází k vytvoření proměnné *adi datového typu tApplicationDeviceInterface. Následující příkaz *adi = adi_GetApplicationInterface();*

přiřazuje našemu ukazateli adresu jedné jediné instance výše zmíněné struktury. Funkce *adi->Init()* nastavuje interní datovou strukturu. Jakmile je práce dokončena, je nutno zavolat *adi->Exit()*

adi->ScanDevices() vrací jména knihoven uložených ve složce "/usr/lib/dal". Tyto knihovny obsahují samotné funkce pro standardy, se kterýma PLC dokáže komunikovat a které jsem popisoval v teoretickém rozboru. Pokud víme, že v našem programu budeme využívat pouze jeden standard - například KBus - je možné ostatní knihovny vymazat. *adi->GetDeviceList* vrací pole struktury tDeviceInfo, která obsahuje unikátní ID zařízení a jejich jméno uložené ve stringu. Pod zařízením si představujme knihovny, které jsou uloženy ve výše zmíněné složce "/usr/lib/dal". K jednotlivým zařízením se přistupuje skrze jejich ID ("DeviceId"). Pokud nedojde k nalezení žádného zařízení, program se ukončí.[2]

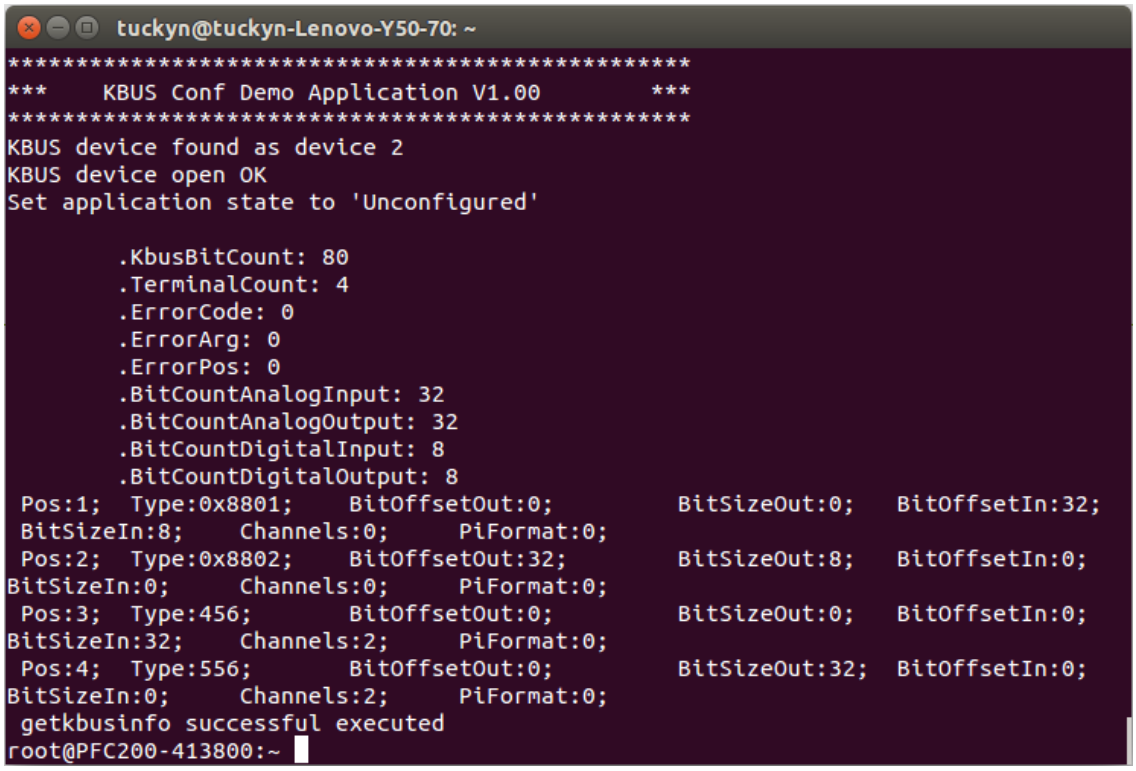
Každý vstupní/výstupní modul je definován přesným množstvím dat a informacemi o konfiguraci. Tato data mohou mít od 2 bitů až do 24 bytů. V závislosti na tom, o jaký modul se jedná, to mohou být vstupní procesní data nebo výstupní procesní data. Použitím funkce *adi->OpenDevice* dojde k identifikaci všech připojených modulů a k vytvoření procesního obrazu vstupů a výstupů. KBus funguje jako posuvný registr operující v cyklech (tzv. "kbus-cycle"). Během jednoho tohoto cyklu dojde k výměně dat mezi každým vstupním modulem a jejich procesním obrazem. Pokud se nepodaří přistoupit ke konkrétnímu modulu, program se opět ukončí.[2]

KBus má dvojí možnost operativního stavu[2]:

- UNCONFIGURED
  - Jiným názvem také automatický operativní stav. KBus je řízen interním vláknem s normální prioritou, který periodicky spouští jednotlivé KBus cykly ("kbus-cycle"). Tento stav je méně časově přesný než RUNNING, avšak je jednoduchý na ovládání a pro náš učel je vhodný.
- RUNNING
  - V tomto stavu (jinak řečeno manuálním) je volání KBus cyklů podmíněno naší aplikací. Vyžaduje se tedy více kódu a komplexnější znalost ADI, naproti tomu se však program stane více deterministický.

Našemu programu tedy přiřazujeme UNCONFIGURED stav pomocí *event.State = ApplicationState_Unconfigured*. Pokud by náhodou při tomto přiřazení došlo k chybě (*adi->ApplicationStateChanged(event)* vrátí chybu), program se opět ukončí.

Abychom zjistili všechny potřebné informace o modulech připojených k našemu PLC, byla použita aplikace *getkbusinfo* dodávaná výrobcem.



Obr. 2.5: Výpis programu getkbusinfo

Z výpisu programu getkbusinfo můžeme vyčíst pozice a modelová čísla jednotlivých modulů. Pro nás je důležitá hlavně informace BitOffset, která nám určuje místo v paměti pro jednotlivé moduly. Vstupní a výstupní moduly mají každý svůj vlastní offset, navzájem se nekříží.

Ke čtení digitálních vstupů je využívaná funkce *adi->ReadBool* běžící v cyklu. DigReadOffset nám určuje přesný bit, který čteme a tento je následně ukládán do proměnné DigRead. Analogové vstupy jsou čteny jako celé byty v paměti pomocí *adi->ReadByte* a jsou ukládány do proměnné AnRead podle kanálu, ze kterého čteme. DigRead a AnRead jsou posléze ukládány do souboru *data_read.json*

Výpis 2.2: Cyklus čtení vstupních modulů

```
adi->ReadStart(kbusDeviceId, taskId);
    for (k = 0; k < 8; k++)
    {
            adi->ReadBool(kbusDeviceId,taskId,DigReadOffset,(_Bool
                *) &DigRead[k]);
      DigReadOffset++;
    }
    adi->ReadBytes(kbusDeviceId, taskId, 0, 2, (uint8_t *) &
        AnRead_ch1);
    adi->ReadBytes(kbusDeviceId,taskId,2,2, (uint8_t *) &AnRead_ch2)
        ;
    adi->ReadEnd(kbusDeviceId, taskId);
    }
```

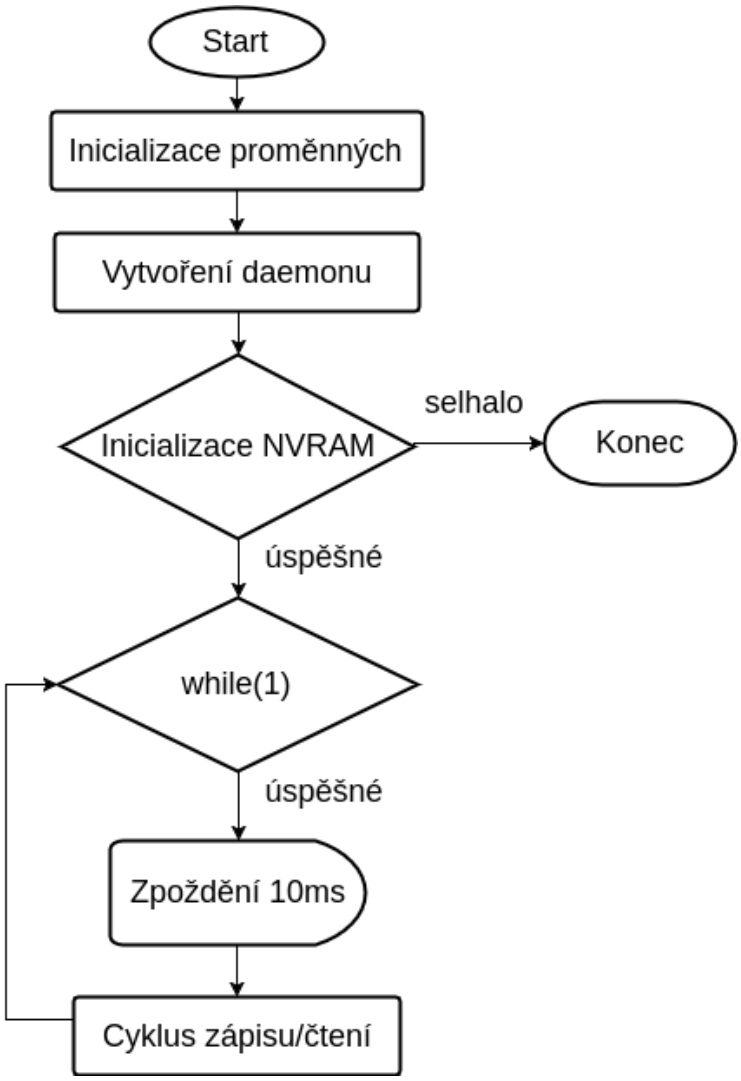
Kód pro zápis se již tak neliší od toho pro čtení. Hodnoty, které chce uživatel zapsat na výstupy, jsou čteny ze souboru *data_write.json* a ukládány do proměnných DigWrite a AnWrite. Pomocí funkcí *adi->WriteBool* a *adi->WriteByte* jsou poté zapisovány na samotné výstupy.

Výpis 2.3: Cyklus zápisu na výstupní moduly

```
adi->WriteStart(kbusDeviceId, taskId);
    for (j = 0; j < 8; j++)
    {
      adi->WriteBool(kbusDeviceId,taskId,DigWriteOffset,DigWrite[j])
          ;
      DigWriteOffset++;
    }
    adi->WriteBytes(kbusDeviceId,taskId,0,2, (uint8_t *) &
        AnWrite_ch1);
    adi->WriteBytes(kbusDeviceId,taskId,2,2, (uint8_t *) &
        AnWrite_ch2);
    adi->WriteEnd(kbusDeviceId,taskId);
```

2.3.2 Variables daemon

Daemon pro ukládání proměnných je o něco jednodušší. Není potřeba, aby využíval funkce z ADI, ale stačí mu využití funkcí přítomných v knihovně *nvram.h*. Přistupuje totiž do NVRAM paměti (Non-Volatile RAM), která si ponechává své hodnoty i přes vypnutí PLC.



Obr. 2.6: Diagram daemon programu pro zápis/čtení proměnných

Výpis 2.4: Struktura plc_variables

```
typedef struct
{
    int var_bool;
    short var_int;
    unsigned short var_word;
    float var_real;
    char var_string[16];
}plc_variables;
```

V programu je definována struktura *plc_variables*, která obsahuje několik datových typů využívaných v programování PLC. *var_bool* zastupuje datový typ bo-

olean, *var_int* short integer, *var_word* word (unsigned short int), *var_real* real (float) a *var_string* string.

Výpis 2.5: Inicializace NVRAM

```
if (nvram_init() < 0)
  {
    return 2;
  }
  else
  {
    nvram_pointer = nvram_get_mapping();
  }
  plc_vars = (plc_variables*)nvram_pointer;
```

nvram_init() inicializuje přístup do NVRAM a *nvram_get_mapping()* vrací pointer na počáteční místo této paměti. Následně je tento pointer přiřazen instanci naší struktury *plc_vars*. Před vstupem do nekonečné smyčky jsou načteny proměnné z paměti a uložené do obou souborů var_*read.json* a *var_write.json*, aby nedošlo ke ztrátě původních dat. Po ukončení všech prací v NVRAM je nutno zavolat *nvram_close()*.

Výpis 2.6: Cyklus zápisu a čtení NVRAM

```
fscanf(var_write, "{\"bool\":%d,\"int\":%hd,\"word\":%hu,\"float\":%
    f,\"string\":\" %s \"}",
                  &plc_vars->var_bool,
                &plc_vars->var_int,
                &plc_vars->var_word,
                    &plc_vars->var_real,
       &plc_vars->var_string);

     fprintf(var_read, "{\"bool\":%d,\"int\":%hd,\"word\":%u,\"float
        \":%f,\"string\":\" %s \"}",
                    plc_vars->var_bool,
                    plc_vars->var_int,
                    plc_vars->var_word,
                    plc_vars->var_real,
       plc_vars->var_string);
```

Ve smyčce je posléze čteno ze souboru *var_write*, tyto hodnoty jsou zapsány do naší struktury a tedy i do paměti NVRAM. Následně je z této paměti čteno a

zapisováno do souboru *var_read*.

2.4 Webové rozhraní

2.4.1 Nastavení Lighttpd

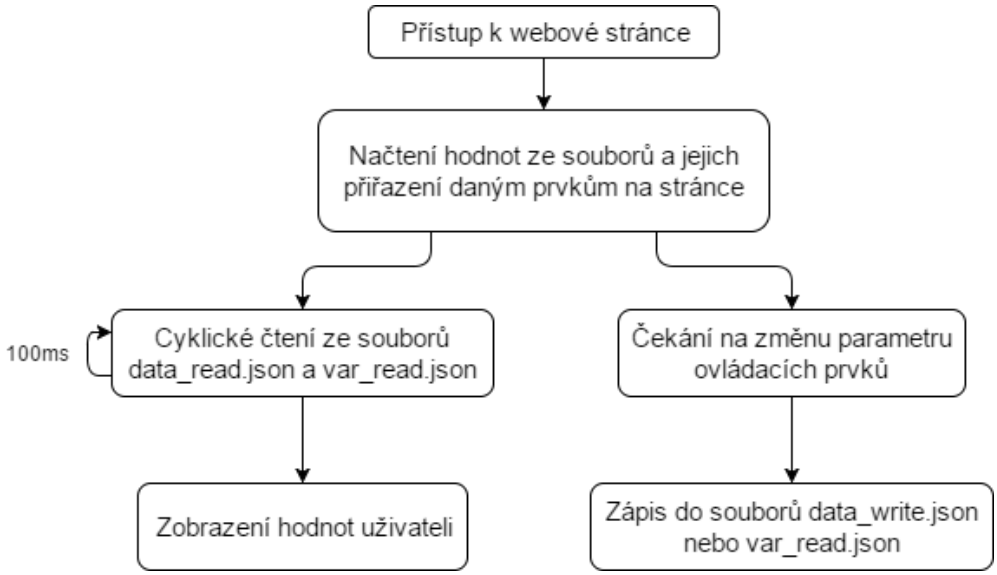
Přímými zásahy do JSON souborů už můžeme měnit hodnoty proměnných nebo I/O modulů. Nyní je na čase propojit tyto soubory s webovou aplikací. Nejdříve však musíme nastavit náš webový server Lighttpd tak, aby dokázal spouštět skripty jako root, jinak nemůžeme přistupovat k ADI. Webový server vystupuje v Pengutronixu jako uživatel "www". Editací souboru *sudoers* tomuto uživateli přiřadíme potřebná práva pro spouštění našich skriptů *json_var_write.php* a *json_write.php*.

Výpis 2.7: Výpis ze souboru sudoers

```
.
.
# User privilege specification
root  ALL=(ALL) SETENV: ALL
admin ALL=NOPASSWD: /etc/config-tools/get_user_info
user ALL=NOPASSWD: /etc/config-tools/get_user_info
www ALL=NOPASSWD: /var/www/testpage/json_write.php, /var/www/
   testpage/json_var_write.php, /sbin/rmmod, /sbin/modprobe,
   /sbin/ifup, /sbin/ifdown, ...
.
.
```

Také je potřeba Lighttpd v konfiguračním souboru povolit potřebné moduly *mod_cgi* a *mod_fastcgi*

2.4.2 Návrh webové aplikace



Obr. 2.7: Diagram funkce webové aplikace

Diagram popisuje funkci webové aplikace. Jakmile dojde na webový server externí požadavek uživatele, dojde k načtení prvotních hodnot ze souborů *var_read.json* a *data_read.json* pomocí funkce *.getJSON()* a jejich následné přiřazení ovládacím a zobrazovacím prvkům stránky. Důvod je prostý, aby mohl uživatel vidět i aktuální zapsané hodnoty na výstupech a nedocházelo tak jejich změnám při každém načtení stránky. Aby nedocházelo k načtení stránky dříve, než Javascript získá hodnoty, máme toto načtení uloženo v podmínce *$(document).ready(function()*, která čeká, až bude stránka načtena.

Následně dochází k cyklickému čtení ze souborů *data_read.json* a *var_read.json* a paralelně k tomuto čtení čekají stránky na event *.change()* ovládacích prvků pro zápis do souborů *data_write.json* a *var_write.json*.

Čtení hodnot ze souborů pomocí JavaScriptu

Následující kód zabezpečuje cyklické čtení hodnot pomocí funkce *.setInterval(,100)*, která zabezpečuje volání celého tohoto kódu každých 100 ms. Aby mohl Javascript načítat ze souboru, je nutné volání asynchronního požadavku *.getJSON()*. Tato metoda/funknce je součástí knihovny JQuery a načítá JSON hodnoty ze souboru použitím GET HTTP požadavku. Jednotlivé elementy JSON souborů jsou poté přiřazeny proměnným a zobrazovány na stránkách ať už textově, nebo graficky.

Výpis 2.8: Cyklické čtení ze souborů JSON

```
window.setInterval(function(){

    $.getJSON("data_read.json", function(data_json) {
      document.getElementById("demo").innerHTML = "Analog input
          channel 1: " + data_json.an_read1 +
      " Analog input channel 2: " + data_json.an_read2 +
      " Digital inputs: " + data_json.dig_read;
      gauge_ch1.SonicGauge ('val', ((data_json.an_read1/3276).
          toFixed(2)));
      gauge_ch2.SonicGauge ('val', ((data_json.an_read2/3276).
          toFixed(2)));
      document.getElementById("digr0").value = data_json.dig_read
          [0];
      document.getElementById("digr1").value = data_json.dig_read
          [1];
      document.getElementById("digr2").value = data_json.dig_read
          [2];
      document.getElementById("digr3").value = data_json.dig_read
          [3];
      document.getElementById("digr4").value = data_json.dig_read
          [4];
      document.getElementById("digr5").value = data_json.dig_read
          [5];
      document.getElementById("digr6").value = data_json.dig_read
          [6];
      document.getElementById("digr7").value = data_json.dig_read
          [7];
    });
    $.getJSON("var_read.json", function(var_json) {
      document.getElementById("demo2").innerHTML = "Bool: " +
          var_json.bool + " Int: " + var_json.int + " Word: " +
          var_json.word + " Real: " + var_json.float + " String: " +
           var_json.string;
    });

var time = new Date();
document.getElementById("demo3").innerHTML = "Actual time: " + time.
    getHours() + ":" + time.getMinutes() + ":" + time.getSeconds() +
     ":" + time.getMilliseconds();

}, 100);
```

Stránka také zobrazuje aktuální čas PLC s přesností na milisekundy z toho důvodu, abychom mohli vidět rychlost obnovování hodnot.

Zápis hodnot do souborů pomocí JavaScriptu a PHP

Aplikace čeká na event od uživatele - změnu hodnot ovládacích prvků pomocí *.change()*. Popíšeme si tuto funkci v následujícím příkladu. "#anwr1"je HTML input umožňující měnit výstupní napětí na prvním kanálu modulu analogového výstupu. Jakmile uživatel změní tuto hodnotu, Javascript zareaguje a zavolá funkci *json_var_write_function()*, která má na starost zapisování hodnot proměnných do JSON souborů.

Výpis 2.9: Reakce JavaScriptu na změnu hodnoty

```
$(document).ready(function(){
    $("#anwr1").change(function(){
        json_var_write_function();
    });
```

Výpis 2.10: Funkce pro zápis hodnot do souborů JSON

```
function json_var_write_function(){
var a = parseInt(document.getElementById("var_bool").value,10);
var b = parseInt(document.getElementById("var_int").value,10);
var c = parseInt(document.getElementById("var_word").value,10);
var d = parseFloat(document.getElementById("var_real").value,10);
var e = " " + document.getElementById("var_string").value + " ";
    var var_json_make = JSON.stringify({"bool":a,"int":b,"word":c,"
        float":d,"string":e});
    $.ajax
    ({
        type: "GET",
        dataType : 'json',
        async: false,
        url: 'json_var_write.php',
        data: { data: var_json_make },
        success: function () {alert("Thanks!"); },
        failure: function() {alert("Error!");}
    });
};
```

Funkce *json_var_write_function()* zapezpečuje zapisování hodnot proměnných do souboru *var_write.json*, který je následně cyklicky čtený Variable daemonem. Na stejném principu funguje také funkce *json_write_function()*, která zapisuje hodnoty

výstupů zvolené uživatelem do souboru *data_write.json*. Obě funkce tyto hodnoty získají při reakci na jejich změnu a poté volají GET - asynchronní požadavek na webový server, který předává data PHP skriptům (*json_write.php* zapisuje výstupy a *json_var_write.php* proměnné), jenž provádí samotnou akci zápisu do souboru. Výpis PHP skriptu je níže.

Výpis 2.11: PHP skript pro zápis do souborů JSON

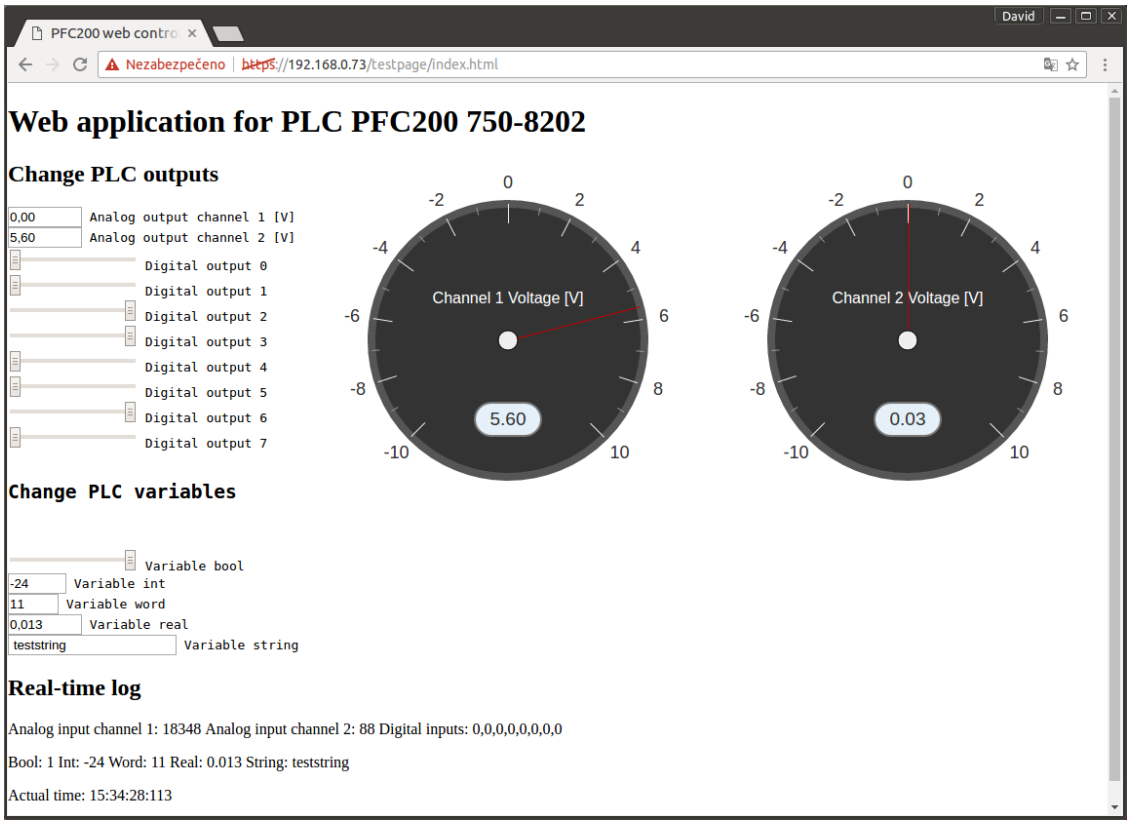
```
<?php
$myFile = "var_write.json";
$fh = fopen($myFile, 'w') or die("can't open file");
$stringData = $_GET["data"];
fwrite($fh, $stringData);
fclose($fh)
?>
```

Funkce pro zápis výstupních hodnot v JavaScriptu i PHP skripty mají podobný zápis jako výše zmíněné, jen pracují s jinými proměnnými.

Výpis 2.12: Syntaxe JSON souborů

```
//data_read.json
{"an_read1":72,"an_read2":88,"dig_read":[0,0,0,0,0,0,0,0]}
 //data_write.json
{"an_write1":0,"an_write2":0,"dig_write":[0,0,0,0,0,0,0,0]}
 //var_read.json
{"bool":1,"int":-24,"word":1,"float":0.000000,"string":" string "}
//var_write.json
{"bool":1,"int":-24,"word":1,"float":0.000000,"string":" string "}
```

Na obrázku 2.8 vidíme výslednou webovou aplikaci. Zobrazení analogových vstupů je realizováno pomocí měřidel a digitálních pomocí výpisu v real-time logu. Ovládání výstupů je vykonáváno v části *Change PLC outputs* pomocí HTML input prvků - pro analogové výstupy přímé zadávání napětí, které je posléze přepočítáváno, a pro digitální pomocí prvku *range*, který má pevně nastavenou hodnotu 0 až 1 (false nebo true). Funguje tedy jako switch. Změna proměnných se provádí v části *Change PLC variables* opět pomocí HTML input prvků s pevně nastavenými hodnotami podle rozsahu jejich datového typu tak, aby nedocházelo k nevhodným zásahům do paměti. Všechny tyto prvky - vstupy/výstupy a proměnné - jsou vyobrazovány v *Real-time log* pro kontrolu, zda došlo k jejich správné změně.



Obr. 2.8: Dynamické webové stránky pro ovládání PLC

2.5 Ověření funkčnosti webové aplikace a možnosti úprav

Funkčnost webové aplikace byla ověřena ve třech webových prohlížečích - Internet Explorer, Google Chrome a Mozzila Firefox. V žádném z nich nebyly patrné nedostatky vyplývající z konkrétního prohlížeče. Při vstupu na stránky došlo k nastavení hodnot dle souborů. Čas a hodnoty ve spodní části stránky se aktualizovaly co 100 ms, což odpovídá našemu nastavení cyklu čtení. Všechny vstupy reagovaly na změnu hodnoty a obecně webová aplikace fungovala podle návrhu.

Řešení, které jsme aplikovali v této práci, je otevřeno mnoha možnostem vylepšení. Výsledek práce operuje s využitím JSON souborů, ke kterým přistupuje jak naše webová aplikace, tak daemon programy. Rozdělení na čtecí a zapisovací soubory je z důvodu vyrušení tzv. race conditions, kdy by mohlo docházet k situacím, že dvě aplikace provádějí zároveň čtení nebo zápis. Nemohli bychom tedy dosáhnout dostatečné stability. Místo těchto JSON souborů je možné do budoucna využít tzv. sockety, které operují přímo v operační paměti a jejich přenos je daleko rychlejší,

než tento zápis/čtení ze souborů. Navíc se tím eliminují race conditions. Vzhledem k tomu, že uživatel nepocítí toto zpomalení a problémy spojené s využitím souboru byly vyřešeny, není tato změna až natolik zásadní.

Kódy programů a webových stránek nejsou až tak robustní, jak by mohly být. Může docházet k problému, když na stránky přistupuje více než jeden uživatel. Tento problém je možné vyřešit použitím loginu na stránkách, případně změně způsobu řešení a přidání dodatečného kódu jak ve webové aplikaci, tak v daemon programech, který by tyto chybové stavy řešil.

Co se týče vzhledu webových stránek je možné jej obohatit o více grafických prvků podobných našim měřidlům.

3 ZÁVĚR

Na programátory a techniky jsou v automatizaci kladeny stále větší nároky a celý obor na sebe pomyslně nabaluje stále více a více nových technologií a způsobů řešení. Názorně to můžeme pozorovat na zdánlivě jednoduše vypadajícím zadání této bakalářské práce, ve které se pojednávalo o alternativním způsobu ovládání skrze webovou aplikaci.

Při práci s embedded Linux prostředím - Pengutronixem - bylo potřeba se detailně seznámit se způsobem fungování tohoto systému. Nástroj PTXdist se snaží působit jednoduše, avšak vyznat se v jeho architektuře a vším tím, co využívá ke svému fungování, zabralo poměrnou dávku času. Dokumentace totiž nepopisuje věci do takového detailu, jaký by byl možná potřeba.

Provést rešerši webových serverů pro tento systém již bylo přijatelnější, jelikož je dostupných spoustu zdrojů a testů, ze kterých může uživatel/student čerpat. Naštěstí pro embedded systémy již existují webové servery vyvíjené přímo pro tyto platformy, které cílí na nízkou náročnost na systém a implementaci relativně moderních technologií. Z naší rešerše nám vyšel Lighttpd jakožto nejvhodnější kandidát, ať už z toho důvodu, že je od WAGA již v PLC dodáván, nebo z důvodu jeho vyladěnosti a nízké náročnosti pro tyto embedded systémy.

Hardwarová abstraktní vrstva DAL, hlavně tedy její rozhraní ADI, je skvělý prostředník pro programátory, umožňující ovládání hardwarových prvků, které mohlo být v minulosti zdlouhavé a zbytečně složité. Umožňuje komunikaci se zařízeními skrze nejvyužívanější PLC standardy a zároveň je také kombinovat. Pro daemon programy se tedy jedná o nezbytnou pomůcku, která nám umožňuje ovládat vstupní a výstupní moduly PLC. Co se týče NVRAM, její ovládání je řešeno pomocí knihovny nvram.h, která obsahuje vše důležité pro zápis/čtení z této paměti.

Samotná webová aplikace podle návrhu funguje a splňuje zadání této práce. V reálném čase reaguje na podněty uživatele nebo PLC. Teoretické zpoždění, ke kterému může během komunikace dojít, je 110 ms (webová aplikace co 100 ms načítá hodnoty a daemon pracuje se zpožděním 10 ms), což nám pro toto zadání stačí a uživatel, ovládající PLC skrze tuto aplikaci, nepocítí nějaké nepříjemné zpoždění. Do budoucna je však vhodné tuto aplikaci udělat více robustní a vylepšit její grafické rozhraní, aby bylo uživatelsky příjemnější.

LITERATURA

[1] WAGO Kontakttechnik GmbH & Co. KG: *WAGO-I/O-SYSTEM 750 Manual*. 2016.

[2] WAGO Kontakttechnik GmbH & Co. KG: *ADI/DAL for PFC*. 2016.

[3] Pengutronix e. K.: *How to become a PTXdist Guru Based on the OSELAS.BSP() Pengutronix Generic-arm*. Hildesheim, 2014.

[4] LinuxPlanet. The IT Business Edge Network. *6 Excellent Linux/Open Source Web Servers* [online]. 06.12.2010 [cit. 26-12-2016]. Dostupné z: <http://www.linuxplanet.com/linuxplanet/reviews/7239/1>.

[5] LINUX.COM. The Linux Foundation. *Which Light Weight, Open Source Web Server is Right for You?* [online]. 04.02.2015 [cit. 26-12-2016].Dostupné z: <https://www.linux.com/news/which-light-weight-open-source-web-server-right-you>.

[6] RootUsers. RootUsers. *Linux Web Server Performance Benchmark – 2016 Results* [online]. 09.03.2016 [cit. 26-12-2016]. Dostupné z: <https://www.rootusers.com/linux-web-server-performance-benchmark-2016-results/>.

[7] Techopedia, Techopedia Inc. *Runtime System* [online]. [cit. 20-05-2017]. Dostupné z: <https://www.techopedia.com/definition/24023/runtime-system>.

[8] Netzmafia, Devin Watson *Linux Daemon Writing HOWTO* [online]. 05.2004 [cit. 20-05-2017]. Dostupné z: <http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html>.

[9] Your HTML source, Ross Shannon *What is HTML?* [online]. 21.08.2012 [cit. 20-05-2017]. Dostupné z: <http://www.yourhtmlsource.com/starthere/whatishtml.html>.

[10] php.net, The PHP Group *What is PHP?* [online]. [cit. 20-05-2017]. Dostupné z: <http://php.net/manual/en/intro-whatis.php>.

[11] MDN Mozilla Developer Network, Mozilla Developer Network and individual contributors *What is JavaScript?* [online]. [cit. 20-05-2017]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript>.

[12] Sally, sallyx.org *CGI* [online]. 20.11.2015 [cit. 20-05-2017]. Dostupné z: <http://www.sallyx.org/sally/c/linux/cgi>.

[13] Sally, sallyx.org *FastCGI* [online]. 31.3.2017 [cit. 20-05-2017]. Dostupné z: <http://www.sallyx.org/sally/c/linux/fcgi>.

[14] Tutorials point *AJAX - Quick guide* [online]. 20.11.2015 [cit. 23-05-2017]. Dostupné z: <https://www.tutorialspoint.com/ajax/ajax_quick_guide.htm>.

A OBSAH PŘILOŽENÉHO CD

- `/` ..... Kořenový adresář přiloženého CD
  - `Bakalářská práce - Webová aplikace v PLC PFC200.pdf`
  - `Daemon programy` ..... Daemon programy pro ovládání HW
    - `bachelor_io_daemon.c` ..... Daemon pro I/O moduly
    - `bachelor_var_daemon.c` ..... Daemon pro NVRAM
  - `Webová aplikace`
    - `testpage` ..... obsahuje všechny soubory pro webovou aplikaci